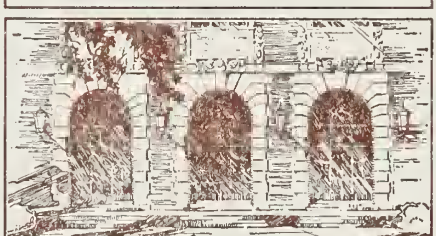

Center for Advanced Computation

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN
URBANA, ILLINOIS 61801

CAC Document No. 56

AUGMENTED SIGNIFICANCE ROUTINES FOR ILLIAC IV

By

R. J. Lermit and J. M. Randal

December 1972

CAC Document No. 56

AUGMENTED SIGNIFICANCE ROUTINES FOR ILLIAC IV

By

R. J. Lermit

and

J. M. Randal

Center for Advanced Computation
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

December 1972

This work was supported in part by the Advanced Research Projects Agency of the Department of Defense and was monitored by the U. S. Army Research Office-Durham under Contract No. DAHC04 72-C-0001; and under Contract No. USAF 30(602)-4144, Order No. 788, implemented initially by the Rome Air Development Center, Research and Technology Division, Air Force Systems Command, Griffiss Air Force Base, New York, and beginning January 17, 1972, by the NASA Ames Research Center, Moffett Field, California.

ABSTRACT

A set of macros to enable users to retain numerical significance during critical phases of a calculation is presented along with the philosophy behind their conception. The macros are designed for speed and have an average accuracy of at least 92 binary places.

TABLE OF CONTENTS

1. INTRODUCTION

These augmented precision algorithms for ILLIAC IV are presented with two aims in mind:

- To provide the general user with a method of keeping numerical significance (and therefore accuracy) even though he is dealing with numbers of very disperate size.

- To record these augmented significance algorithms so that they may be modified by future ILLIAC IV.

Their function is thus different from the algorithms described by Yasui [1] in the sense that they do not maintain 96 binary places of accuracy, but only the accuracy desired by the user up to about 92 places. As a consequence, they are faster and less bulky than full double precision routines, and use neither CU instructions nor bits of the RGD. The CU and mode bits are thus left free to control the main algorithm.

Because these algorithms are directed toward maintaining significance, routines are also provided for obtaining augmented numbers from sums and products of single precision numbers. A routine is provided for rounding augmented precision numbers and converting them to single precision.

No facilities are provided for reading or printing augmented precision numbers. Indeed it is envisioned that users will use them only temporarily in their programs to ensure accuracy during certain critical parts of a calculation, and convert augmented precision results to single precision before reaching the end of their algorithm.

The initiated reader may observe that more elegant macros may be achieved by using "conditional compile" facilities, but since the

paper is written for the general user, our aim is clarity at the expense of elegance whenever the latter would make the method less comprehensible.

2. REPRESENTATION

A row of augmented precision numbers is represented by two rows of ordinary ILLIAC IV 64-bit floating point numbers. [One word (a^h); the more significant part, is not necessarily normalized, but its exponent is the true exponent for the whole augmented number. The other word (a^l) is the less significant or adjustment part of the number. Its exponent is usually about 48 less than the exponent of a^h, the mantissa may be of different sign from that of a^h, but when the mantissa of a^l is aligned so that it is of the same sign as a^h and its exponent is exactly 48 less than the exponent of a^h, then $(a^h, a^l) = a^h + a^l$ represents a double precision number with the exponent of a^h.

However, this "aligned" representation is rarely necessary, and is not explicitly maintained by the augmented precision algorithms. It is the relaxation of these rules of representation that give these algorithms their speed and compactness with a small cost in accuracy.

Throughout this document a^h and a^l are represented as AH and AL respectively, in program text, the terminal H or L representing the more or less significant part of the number respectively. Single precision numbers are represented by single letters without an H or L ending.

A single precision number x, is expressed as (x, 0) in augmented precision.

An augmented precision number has a value within a neighborhood ε of a single precision number if the result of normalizing the difference between the augmented and single precision number has a modulus less than ε. A similar relationship holds between augmented precision numbers.

3. USE

All augmented arithmetic algorithms presented in this document are represented as defines which, except for a few cases, operate on an augmented precision accumulator represented by the labels ACC and AUXACC. ACC and AUXACC contain the more and less significant part of the augmented result. The accumulator is merely a device to reduce the length of parameter bits in the defines, and because the need for augmented precision numbers tends to arise from the need to accumulate sums and products of single precision numbers, saves unnecessary access times as well.

The parameters of the defines may be any valid PE address allowable in ASK [2] providing they do not conflict with the register requirements of the defines being invoked (See Appendix B).

Augmented precision values may be created by:

1. Assigning ACC and AUXACC the augmented precision value directly:

```
LIT(3) = 1.0;
LDA $C3;
STA ACC;
CLAR;
STA AUXACC;
```

(or more directly:

```
LIT(3) = 1.0;
DLOAD $C3 = 0;)
```

will create the augmented precision number (1.0, 0) in the accumulator.

2. Using the ILLIAC IV extended add (EAD) [3] or extended subtract (ESB) instructions,

```
DEFINE AUX &A &B =
```

```
LDA &A;
EAD &B;
STB ACC;
STA AUXACC ##;
DEFINE SUX &A &B =
LDA &A;
ESB &S;
STB ACC;
STA AUXACC ##;
```

will form augmented precision value in the accumulator from the sum or difference respectively of invocation values of &A and &B.

3. Using the MUX define to form the product of two single precision numbers. Because the ILLIAC IV multiply (ML) instruction does not produce a true augmented precision product, the define DML is provided to do so (see Appendix C). The instructions:

```
LDA X;
DML Y;
```

leave the more significant part of the augmented precision product of X and Y in RGR and the less significant part in RGA. Thus:

```
DEFINE MUX &A &B =
LDA &S;
DML &B;
STR ACC;
STA AUXACC ##;
```

The defines OCLEAR, DLOAD and DSTORE (which respectively, clear, load and store the augmented precision accumulator) are provided so that the user may work with an augmented ILLIAC IV which in addition to the

existing ILLIAC IV facilities has a single augmented precision accumulator upon which all augmented precision functions act.

This augmented precision accumulator destroys the symmetric properties of the augmented precision operations in the sense that it alone is normalized during the augmented precision operations. It is truly an accumulator of sums or products, and its use in this role automatically increases the accuracy of the calculation (See Section 5). To dispense with a unique accumulator would either increase the execution times of the augmented arithmetic functions or reduce their accuracy.

Generally speaking, the results of augmented arithmetic calculation are not normalized after the calculation because they will be normalized during the next. However, they are normalized before being converted back into single precision numbers.

4. FUNCTIONS

In addition to the basic functions described in the last section, the package provides the following facilities:

4.1 Operations involving single precision numbers.

DEFINE DOT &A &B

This operation forms the double length product of &A and &B and adds it to the augmented precision accumulator. Its repeated application forms the scaler product of its successive augments.

DEFINE DPLUS &A

This operation adds the single precision quantity &A to the augmented precision accumulator.

DEFINE DMINUS &A

This operation subtracts the single precision quantity &A from the augmented precision accumulator.

DEFINE DTIMES &A

This function multiplies the augmented precision accumulator by the single precision number &A. The result is left in the augmented precision accumulator.

4.2 Operations on the augmented precision accumulator.

DEFINE DNEG

This operation negates the augmented precision accumulator.

DEFINE DRECIP

This operation replaces the contents of the augmented precision accumulator by its reciprocal. The method used includes finding the reciprocal of the more significant part of the accumulator and refining this with one application of Newton's method for reciprocation. No other facilities for augmented precision division are provided in the package.

DEFINE DNORM

This operation normalizes the augmented precision accumulator by normalizing its more significant half and adding its less significant half to the result. This process is repeated once in case the more significant part was originally zero and the less significant part was unnormalized.

DEFINE SINGLE

This operation normalizes the augmented precision accumulator and then rounds it by adding a quantity of the correct sign and exponent. The single precision result is left in RGA.

4.3 Operations involving double precision numbers.

DEFINE DADD &AH &AL

The augmented precision number (&AH, &AL) is added to the augmented precision accumulator.

DEFINE DSUB &AH &AL

The augmented precision number (&AH, &AL) is subtracted from the augmented precision accumulator.

DEFINE DMULT &AH &AL

The augmented precision accumulator is multiplied by the augmented precision number (&AH, &AL). The method used is equivalent to the following algorithm, except that only the more significant part of MUX &AL &YL is generated and used.

```
DEFINE DMULT &AH &AL =
DSTORE YH YL;
MUX &AL YL;
DOT &AL YH;
DOT &AH YL;
DOT &AH YH ##;
```

5. ACCURACY

Since the extended precision hardware on ILLIAC IV differs from that pprovided on most machines, it is necessary to consider how this affects the accuracy of the result obtained.

Analysis will be confined to the accumulation of the inner product of two vectors, $\sum_{i=1}^{n} a_i b_i$. This is, by far, the most common use of double precision arithmetic.

It is assumed that the calculations are being performed serially; i.e., partial sums are being calculated--$S_i = \sum_{j=1}^{i} a_j b_j$ from: $S_i = S_{i-1} + a_i b_i$. Normally, 64 such inner products would be accumulated simultaneously. However, if summation is being carried out across the PEs by a method such as the "log sum" [6] technique, the bound on the error is even smaller, (See Linz [5]). The method analyzed is, therefore, "worst case".

Consider first the accumulation of inner products on a standard computer using a word with t bits mantissa and accumulating products in a double length register of 2t bits. The final sum is then rounded to single precision.

Let fl(E) represent the evaluation in floating point arithmetic of the expression E. If S_i is the i^{th} partial sum, then

$$S_o = 0$$

$$S_i = fl(s_{i-1} + fl\ (a_i x\ b_i))$$

$$i = 1, 2, \ldots, n.$$

The product of the single precision operands a_i, b_i is first formed, giving a double length product; this calculation is exact. Adding this product to the partial sum S_{i-1}, using double length arithmetic, results in a rounding error:

$$S_i = fl(S_{i-1} + fl(a_i \times b_i))$$

$$= fl(S_{i-1} + a_i b_i)$$

$$= S_{i-1} \ (1+\varepsilon_i) + a_i b_i \ (1+\delta_i).$$

It may be shown that:

$$|\varepsilon_i|, \quad |\delta_i| \quad \leq \frac{3}{2} \, 2^{-2t},$$

[See for instance Wilkinson [4]], and hence:

$$fl \ (a^T b) = fl \ (\sum_{i=1}^{n} a_i b_i) = S_n$$

$$= \sum_{i=1}^{n} a_i b_i \ (1+\delta_i) \ (1+\varepsilon_{i+1}) \ (1+\varepsilon_{i+2}) \ \cdots \ (1+\varepsilon_{i+n})$$

$$= \sum_{i=1}^{n} a_i b_i \ (1+\gamma_i)$$

where:

$$(1-\frac{3}{2} \ 2^{-2t})^{n-i+1} \leq 1 + \gamma_i \leq (1 + \frac{3}{2} \, 2^{-2t})^{n-i+1} .$$

Since the factor $(1 \pm \frac{3}{2} \ 2^{-2t})^{n-i+1}$ is inconvenient, we make the assumption that:

$$\frac{3}{2} \ n \ 2^{-2t} < 0.1.$$

This will be true for any value of n found in practice since 2^{-2t} is very small. The inequalities for γ_i may then be simplified to:

$$|\gamma_i| < \frac{3}{2} \, (1.06) \ (n-i+1) \ 2^{-2t}$$

$$\leq \frac{3}{2} \, (1.06) \quad n \ 2^{-2t}.$$

Denoting by $fl_{2,1} \ (a^T b)$ the result of rounding $fl \ (a^T b)$ to single precision, a bound on the absolute error may be found from:

$$|fl_{2,1} \ (a^T b) - a^T b|$$

$$\leq |fl_{2,1} \ (a^T b) - fl \ (a^T b)|$$

$$+ \ |fl(a^T b) - a^T b|$$

$$\leq |fl(a^T b)| \ 2^{-t} + \sum_{i=1}^{n} |a_i b_i \ \gamma_i|$$

$$= |\sum_{i=1}^{n} a_i b_i (1+\gamma_i)| \ 2^{-t} + \sum_{i=1}^{n} |a_i b_i \ \gamma_i|$$

$$\leq |\sum_{i=1}^{n} a_i b_i| \ 2^{-t} + (1+2^{-t}) \sum_{i=1}^{n} |a_i| \ |b_i| \ |\gamma_i|$$

$$\leq 2^{-t} \ |a^T b| + \frac{3}{2} .1.1 \ n \ 2^{-2t} \sum_{i=1}^{n} |a_i| \ |b_i|$$

Therefore, using the Schwarz inequality, the absolute error is bounded by:

$$2^{-t} \ |a^T b| + \frac{3}{2} .1.1 \ n \ 2^{-2t} \ ||a||_2 \ \ ||b||_2 \ .$$

Note that this does not necessarily give a small relative error; however, unless severe cancellation takes place, the second term is negligible compared with the first.

In performing the analogous calculations using ILLIAC IV hardware, the following steps are performed:

5.1 At the i^{th} step, a_i and b_i are multiplied giving a double length product. No round off error is produced.

$$(p_i^h, p_i^l) = fl \ (a_i \times b_i) = a_i b_i$$

It is assumed that the a_i and b_i are normalized. This is essential for the error analysis. [Intuitively we want to push as much of the product into the high precision part as possible.]

5.2 The low precision parts of p_i and S_{i-1} are added using an ordinary rounded and normalized add. Call the result u_i

$$u_i = fl\ (p_i^l + S_{i+1}^l)$$

$$= p_i^l\ (1 + \varepsilon_i) + S_{i-1}^l\ (1 + \varepsilon_{i2}).$$

5.3 S_{i-1}^h is added to p_i^h, using the EAD instruction, forming S_h^i with the overflow going into v_i. Normalizing S_{i-1}^h controls the error. Intuitively we put as much as the sum in the high order word as possible.

$$(S_i^h,\ v_i) = fl(S_{i-1}^h\ @\ p_i^h) = S_{i-1}^h + p_i^h$$

Where @ denotes the EAD operator, no round off error is involved; in fact:

$$S_i^h + v_i = S_{i=1}^h + p_i^h.$$

And hence,

$$S_i^h = \sum_{j=1}^{i} p_j^h - \sum_{j=1}^{i} v_j$$

S_i^l is formed from u: and v_i using a rounded and normalized add; i.e.,

$$S_i^l = fl\ (u_i + v_i)$$

$$= u_i\ (1 + \varepsilon_{i3}) + v_i\ (1 + \varepsilon_{i4})$$

where $|\varepsilon_{i1}|$, $|\varepsilon_{i2}|$, $|\varepsilon_{i3}|$, $|\varepsilon_{i4}|\ \leq 2^{-47}$.

Using the above equations, we have

$$S_i = S_{i-1} + a_i b_i + E_i$$

where $E_i = [p_i^l\ (1 + \varepsilon_{i1}) + S_{i-1}^l\ (1 + \varepsilon_{i2})]\ (1 + \varepsilon_{i3}) + v_i\ (1 + \varepsilon_{i4})$

$$- [p_i^l + S_{i-1}^l + v_i]$$

$$= p_i^l\ [(1 + \varepsilon_{i1})\ (1 + \varepsilon_{i3}) - 1] + v_i\ \varepsilon_{i4} + S_{i-1}^l\ [(1 + \varepsilon_{i2})(1 + \varepsilon_{i3}) - 1].$$

Bounds must be obtained for $|p_i^l|$, $|v_i|$ and $|S_{i-1}^l|$.

$$|p_i^l| \leq 2^{-46}\ |p_i^h| \leq 2^{-46}\ |a_i b_i|$$

This depends on the fact that a_i and b_i are normalized. The mantissa part of p_i^h is at least $\frac{1}{4}$ since that of both a_i and b_i is at least $\frac{1}{2}$. Because the mantissa part of p_i^l is not greater than 1, and the exponents of p_i^l and p_i^h differing by exactly 48. Since it is the low precision part of an extended add of S_{i-1}^h and p_i^h, a bound for $|v_i|$ is

$$|v_i| \leq 2^{-46} \max \{|S_{i-1}^h|, |p_i^h|\} .$$

This follows since the exponent of v_i is at least 48 less than the larger of S_{i-1}^h and p_i^h, and the mantissa part of p_i^h is at least $\frac{1}{4}$ and that of S_{i-1}^h is at least $\frac{1}{2}$ (since it is normalized). Thus,

$$|v_i| \leq 2^{-46} \max \{|\sum_{j=1}^{i-1} p_j^h - \sum_{j=1}^{i-1} v_j|, p_i^h\}$$

$$\leq 2^{-46} \{\sum_{j=1}^{n} |p_j^h| + \sum_{j=1}^{n} |v_j|\}$$

and

$$\sum_{i=1}^{n} |v_i| \leq 2^{-46} n \{\sum_{j=1}^{n} |a_j b_j| + \sum_{j=1}^{n} |v_j|\}$$

$$\sum_{i=1}^{n} |v_i| \{1 - 2^{-46} n\} \leq 2^{-46} n \; ||a||_2 \; ||b||_2 .$$

Hence, for any reasonable n,

$$\sum_{i=1}^{n} |v_i| \leq 1.1 \times 2^{-46} n \; ||a||_2 \; ||b||_2 .$$

It only remains to find a bound for S_i^l.

$$S_i^l = u_i (1 + \varepsilon_{i3}) + v_i (1 + \varepsilon_{i4})$$

$$= p_i^l (1 + \varepsilon_{i1}) (1 + \varepsilon_{i3}) + v_i (1 + \varepsilon_{i4}) + S_{i-1}^l (1 + \varepsilon_{i2}) (1 + \varepsilon_{i3})$$

$$= \sum_{j=1}^{i} p_j^l (1 + \varepsilon_{ji}) (1 + \varepsilon_{j3}) (1 + \varepsilon_{j+1,2}) \times (1 + \varepsilon_{j+1,3}) \cdots (1 + \varepsilon_{i2})$$

$$(1 + \varepsilon_{i3}) + \sum_{j=1}^{i} v_j \, (1 + \varepsilon_{j4}) \, (1 + \varepsilon_{j+1,2}) \, (1 + \varepsilon_{j+1,3}) \ldots$$

$$(1 + \varepsilon_{i2}) \, (1 + \varepsilon_{i3})$$

$$= \sum_{j=1}^{i} p_j^l \, (1 + \gamma_j) + \sum_{j=1}^{i} v_j \, (1 + \delta_j)$$

where, as before, it may be shown that

$$|\gamma_j|, \ |\delta_j| < \frac{3}{2} \, (1.06) \, (2n) \, 2^{-48} = \Delta.$$

However, there may be up to 2n factors in each terms and the work is being done in single precision. Therefore,

$$S_i^l = \sum_{j=1}^{i} p_j^l + \sum_{j=1}^{i} v_i + \sum_{j=1}^{i} p_j^l \gamma_j + \sum_{j=1}^{i} v_j \, \delta_j.$$

But since,

$$S_i^h = \sum_{j=1}^{i} p_j^h - \sum_{j=1}^{i} v_j$$

then,

$$S_i = S_i^h + S_i^l$$

$$= \sum_{j=1}^{i} p_j^h + \sum_{j=1}^{i} p_j^l + \sum_{j=1}^{i} p_j^l \gamma_j + \sum_{j=1}^{i} v_j \, \delta_j$$

$$= \sum_{j=1}^{i} a_j b_j + \sum_{j=1}^{i} p_j^l \gamma_j + \sum_{j=1}^{i} v_j \, \delta_j.$$

Putting $i = n$

$$fl \, (a^T b) = S_n$$

$$= a^T b + \sum_{i=1}^{n} p_i^l \gamma_i + \sum_{i=1}^{n} v_i \, \delta_i$$

The absolute error is, thus:

$$\left| \sum_{i=1}^{n} p_i^l \gamma_i + \sum_{i=1}^{n} v_i \, \gamma_i \right| \le \sum_{i=1}^{n} |p_i^l| \, \Delta + \sum_{i=1}^{n} |v_i| \, \Delta$$

$$\leq 2^{-46} \sum_{i=1}^{n} |a_i b_i| \Delta + (1.1 \times 2^{-46} n \, ||a||_2 \; ||b||_2) \Delta$$

$$\leq 1.1 \times 2^{-46} \; \Delta \; (n + 1) \; ||a||_2 \; ||b||_2$$

$$\leq 1.1 \times 1.06 \times 3 \times 2^{-46} \times 2^{-48} n(n+1) \; ||a||_2 \; ||b||_2$$

$$\leq 2^{-92} \; n(n + 1) \; ||a||_2 \; ||b||_2 .$$

Rounding the result to single precision, as before, and calling the result $fl_{2,1}(a^T b)$, the following bound is obtained:

$$|fl_{2,1}(a^T b) - a^T b| \leq 2^{-48} \; |a^T b| + 2^{-92} \; n(n + 1) \; ||a||_2 \; ||b||_2 .$$

Putting a mantissa size $t = 48$ into the error bound for a standard machine gives, by comparison:

$$2^{-48} \; |a^T b| + 1.1 \; (\tfrac{3}{2} n) \; 2^{-96} \; ||a||_2 \; ||b||_2 .$$

The essential difference between these two results is that the analysis for ILLIAC IV produces a factor of $n(n + 1)$ in the second term rather than a factor of n. Even for large n, however, this term should be negligible unless severe cancellation takes place. ILLIAC IV benefits further from a large mantissa size. The method used is therefore fully justified.

6. SPACE AND TIMING

Instruction times quoted in the ILLIAC IV Systems Characteristics and Programming Manual [3] have been used in estimating the execution times of the appropriate functions. No FINSTAPE overlap has been assumed. One clock memory access time has been added for RGS and 7 clocks for memory. Each define parameter is assumed to imply a memory access except where actual parameters in the package indicate otherwise. Space and execution times are presented in the following table:

FUNCTION	SPACE (syllables)	TIME (PE clocks)
DADD	7	72
DCLEAR	3	17
DLOAD	4	32
DMINUS	6	52
DML	8	26
DMULT	60	379
DNEG	6	36
DNORM	9	63
DOT	17	102
DPLUS	6	52
DRECIP	84	547
DSTORE	4	32
DSUB	7	72
DTIMES	29	161
MUX	11	50
SINGLE	18	98

Table 6.1 Space and Execution Time

Users wishing to convert any of these defines into subroutines should consult Appendix A.

7. CONCLUSION

These augmented precision routines are designed to help the user retain numerical significance during certain critical parts of a calculation rather than to provide double precision routines *per se*. The modest sacrifice of accuracy,an economical number representation, and a certain symmetry in otherwise associative operations are, the authors felt, more than amply repaid by increased execution speed.

It remains to be seen whether the general user agrees with this thesis.

REFERENCES

[1] Yasui, T., Double Precision Algorithms for ILLIAC IV.

[2] ILLIAC IV Assembler, ILLIAC IV Software Reference Manual, Vol. 2, Chapter 1.

[3] ILLIAC IV Programming and Characteristics Manual.

[4] Wilkinson, J. H., The Algebraic Eigenvalue Problem, Oxford, 1965.

[5] Linz, Peter, Accurate Floating-Point Summation, Comm. ACM 13, 6 (June 1970), pp. 361-362.

[6] Denenberg, S. An Introduction to the ILLIAC IV System.

APPENDIX A. Defines as Subroutines

Two defines, DMULT (30 words) and DRECIP (42 words), occupy enough space and execute long enough to be made subroutines (if they are to be invoked more than once) without appreciably degrading their performance. For the purposes of this conversion, the augmented precision functions fall into two classes.

A.1 Defines Without Parameters

Because the augmented precision accumulator is the only operand, the conversion is easy and the subroutine becomes:

```
RECIPS::RECIP;
        EXCHL(3) $ICR;
```

and may be called by invoking the standard CALL define:

```
DEFINE CALL &NAME =
CLC(3)
SLIT(2) = &NAME;
EXCHL(3) $ICR ##;
```

thus:

```
CALL RECIPS;
```

This complies with subroutine standards.

A.2 Defines With Parameters

There are two useful methods.

A.2.1 The user may declare row variables for use when passing parameters to the subroutine. If XH and XL are user declared variables, then the subroutine may look like the following:

```
MULTS::MULT XH XL;
       EXCHL(3) $ICR;
```

(where XH, XL have been declared XH:BLK 1; XL:BLK 1;) and the calling

sequence would become

```
        LDA ARGH;
        STA XH;
        LAD ARGL;
        STA XL;
        CALL MULTS:
```

That is, the subroutine has been made parameterless.

A.2.1. A slightly faster method is to use ACAR0 and ACAR1 to "point" to the correct arguments. The calling sequence would then be:

```
        CLC(0);
        SLIT(0) = ARGH;
        CLC(1);
        SLIT(1) = ARGL;
        CALL MULTSS:
```

where the subroutine is now:

```
        MULTSS::MULT 0(0) 0(1);
                EXCHL(3) $ICR ##;
```

A little more elegance may then be achieved with:

```
        DEFINE EXECUTE &NAME &AH &AL =
        CLC(0);
        SLIT(0) = &AH;
        CLC(1);
        SLIT(1) = &AL;
        CALL &NAME ##;
```

Both methods comply with standard subroutine conventions.

APPENDIX B. Define Dependencies

Some defines invoke others which, in turn, invoke defines. The list below illustrates these dependencies. Invoked defines are followed by a list in parentheses of the defines which they invoke.

DMULT	DSTORE, DOT (DML, DADD)
DOT	DML, DADD
DRECIP	DTIMES (DML), DMINUS, DNEG
DTIMES	DML
MUX	DNORM
SINGLE	DNORM

Two defines use memory locations which must be declared by the user. The memory locations are:

```
DTEMPH:  BLK 1;
DTEMPL:  BLK 1;
```

The defines using those memory locations are:

DMULT	DTEMPH, DTEMPL
DRECIP	DTEMPH

If the user has rows X and Y say, which are available for use by DMULT or DRECIP, the define:

```
DEFINE DTEMPH = X ##, DTEMPL = Y ##;
```

declared before DMULT or DRECIP is invoked will cause them to use X and Y in place of DTEMPH and DTEMPL.

P.E. registers RGR and RGS are used by the following defines:

DMULT	RGR, RGS
DNORM	RGR
DOT	RGR, RGS

DRECIP	RGR, RGS
DTIMES	RGR, RGS
MUX	RGR, RGS
DNORM	RGR

All defines use RGA and RGB. None use RGX.

APPENDIX C.

The annotated bodies of the augmented precision defines appear below. Their use supposes that exponent orderflow does not cause the F-bit to be set.

```
LISTED ON: 10/05/71   AT: 13:04

          DEFINE DML &B =
          ML &B;
          ASB;
          LDR &A;
          LDS &A;
          LDA =3F00:16;
          SHAL 4#;
          ADM &S;
          ADEX &R #*;

          DEFINE DTIMES &A =
          LDA &A;
          DML AUXACC;
          LDS ACC;
          STR ACC;
          STA AUXACC;
          LDA &A;
          DML &S;
          LDS &A;
          DADD &R &S #*;

          DEFINE DPLUS &A =
          LDA ACC;
          NORM;
          EAD &A;
          STB ACC;
          ADR AUXACC;
          STA AUXACC #*;

          DEFINE DMINUS &A =
          LDA ACC;
          NORM;
          ESB &A;
          STB ACC;
          ADR AUXACC;
          STA AUXACC #*;

          DEFINE DNORM =
          LDA ACC;
          NORM;
          EAD AUXACC;
          LDR &A;
          LDA &B;
          NORM;
          EAD &R;
          STB ACC;
          STA AUXACC #*;

          DEFINE DCLEAR =
          CLRA;
          STA ACC;
          STA AUXACC #*;

          DEFINE DLOAD &AH &AL =
          LDA &AH;
          STA ACC;
          LDA &AL;
          STA AUXACC #*;
```

```
DEFINE DSTORE %AH %AL =
LDA ACC;
STA %AH;
LDA AUXACC;
STA %AL ##;

DEFINE DADD %AH %AL =
LDA ACC;
NOR 1;
EAD %AH;
STB ACC;
ADR %AL;
ADR AUXACC;
STA AUXACC ##;

DEFINE DSUB %AH %AL =
LDA ACC;
NOR 1;
ESB %AH;
STB ACC;
 SBR %AL;
ADR AUXACC;
STA AUXACC ##;

DEFINE MUL %A %B =
LDA %A;
DML %B;
STB ACC;
STA AUXACC ##;

DEFINE DOT %A %B =
LDA %A;
DML %B;
LDS %A;
DADD %R %S ##;

DEFINE DMULT %AH %AL =
DSTORE DTEMPH DTEMPL;
LDA %AL;
MLRN DTEMPL;
STA ACC;
LDB =0;
STB AUXACC;
DOT %AH DTEMPL;
DOT %AL DTEMPH;
DOT %AH DTEMPH ##;

DEFINE DNEG =
LDA ACC;
CHSA;
STA ACC;
LDA AUXACC;
CHSA;
STA AUXACC ##;

DEFINE DRECIP =
LDA ACC;
NOR 1;
LDR %A;
LDA =1003:15;
SHAL 47;
```

```
LDB =0;
DVN $R;
STA DTEMP4;
DTIMES DTEMP4;
DTIMES DTEMP4;
DMINUS DTEMP4;
DMINUS DTEMP4;
DNEG ##;

DEFINE SINGLE =
DNOR 1;
LDA =7FA1116;
SHAL 47;
LDB $A;
LDA ACC;
ASB;
SWAP;
ADEX $B;
EAD ACC;
SWAP ##;
```

APPENDIX D. The Use of Conditional Definition.

The reader will notice that the defines DADD and DPLUS differs by one parameter and one ASK instruction. By using the conditional assembly features of ASK [2], DPLUS and DADD may be combined, as can DMINUS and DSUB:

```
DEFINE DADD &AH &AL =
LDA ACC;
NORM;
EAD &AH;
STB ACC;
&IF &EMPTY(&AL) &THEN % IF SECOND PARAMETER GIVEN
&ELSE ADR &AL; &FI& %THEN USE IT
ADR AUXACC;
STA AUXACC ##;

DEFINE DSUB &AH &AL =
LDA ACC;
NORM;
ESB &AH;
STB ACC;
&IF &EMPTY(&AL) &THEN %IF SECOND PARAMETER GIVEN
&ELSE SBR &AL; &FI;   %THEN USE IT
ADR AUXACC;
STA AUXACC ##;
```

The invocation:

```
DADD A;
```

is now identical with the invocation

```
DPLUS A;
```

while the call

```
DADD A, B;
```

retains its original meaning.

The provision of defines with conditional bodies thus makes the macro package easier to use in the sense that fewer define names need to be learned and understood.

It is worth mentioning, however, that combining DTIMES and DMULT is not as neat as the above examples:

```
DEFINE DMULT &AH &AL =
%IF &EMPTY(&AL) &THEN %IF SECOND PARAMETER ABSENT
                      %THEN ISSUE CODE FOR DTIMES
LAD &AH;
DML AUXACC;
LDS ACC;
STR ACC;
STA AUXACC;
LDA &AH;
DML $S;
LDS $A;
DADD $R $S
&ELSE                 %OTHERWISE
DSTORE DTEMPH DTEMPL; %ISSUE CODE FOR DMULT
LDA &AL;
MLRN DTEMPL;
STA ACC;
LDB = 0;
```

```
STB AUXACC;
DOT &AH DTEMPL;
DOT &AH DTEMPH;
DOT &AH DTEMPH   &FI;
```

However, the notational economy makes the use of conditional compilation worthwhile.

Conditional compilation may also be used to increase the scope of the define. For instance, when accumulating sums of positive numbers, one knows the more significant word of the augmented precision accumulator is non-zero, and thus some of the instructions in DNORM are unnecessary. A version of DNORM that may be used to normalize only as far as the more significant half of the augmented precision accumulator or to completely normalize the accumulator might be written as follows:

```
DEFINE DNORM &N =
LDA ACC;
NORM;
EAD AUXACC;
&IF &N &THEN %IF &N IS ODD, NORMALIZE TO ACC ONLY &ELSE
LDR $A;      %OTHERWISE NORMALIZE WHOLE ACCUMULATOR
LDA $B;
NORM;
EAD $R; &FI;
STB ACC;
STA AUXACC ##;
```

Thus, the invocation

```
DNORM 1;
```

normalizes only as far as the more significant half of the augmented precision accumulator, while

```
DNORM 2;
```

normalizes the whole augmented precision accumulator.

Use of conditional compilation facilities may thus enhance the efficiency of the compiled program without the notational disadvantage of having to provide a myriad of specific subroutines or macros for every function variant.

The facilities presented here are mainly illustrative and are not present on the standard library tape. The user, considering the augmented precision macros in the light of his own particular application, will no doubt devise suitable local enhancements.

UNCLASSIFIED

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)* Center for Advanced Computation University of Illinois at Urbana-Champaign Urbana, Illinois 61801	2a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED 2b. GROUP

3. REPORT TITLE

Augmented Significance Routines for ILLIAC IV

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*

Research Report

5. AUTHOR(S) *(First name, middle initial, last name)*

J. M. Randal and R. J. Lermit

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
December 1972	36	6

8a. CONTRACT OR GRANT NO. DAHC04 72-C-0001 and USAF 30(602)4144 b. PROJECT NO. ARPA Order No. 1899 and No. 788 c. d.	9a. ORIGINATOR'S REPORT NUMBER(S) CAC Document No. 56 9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*

10. DISTRIBUTION STATEMENT

Copies may be requested from the address given in (1) above.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY U.S. Army Research Office-Durham Duke Station, Durham, North Carolina and: NASA Ames Research Center, Mail Stop 233-14, Moffett Field, Calif.

13. ABSTRACT

A set of macros to enable users to retain numerical significance during critical phases of a calculation is presented along with the philosophy behind their conception. The macros are designed for speed and have an average accuracy of at least 92 binary places.

DD FORM 1 NOV 65 1473

UNCLASSIFIED

Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Error Analysis, Computer Arithmetic Macro-Assembler						